Computer Network Lab4 Report

吕峻汉 231220021

October 30, 2024

1 ask 2: IP Forwarding Table Lookup

Assignment: In the report, show how you match the destination IP addresses.

Listing 1: Match destination IP address

```python

def match(self, target_ip):
        #longest prefix match
        result = {}
        length = 0
        for entry in self.list_dic:
            concatenate_addr = IPv4Network(f"{entry['network-address']}/{entry['netmask']}")
            #assert target_ip is an object of IPv4Address here
            if(target_ip in concatenate_addr):
                #print(f"ca:{concatenate_addr} ip: {target_ip}")
                if(length < concatenate_addr.prefixlen):
                    length = concatenate_addr.prefixlen #find longest prefix
                    result = entry

        return result

def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
        timestamp, ifaceName, packet = recv
        # TODO: your logic here
        #get possible header
        arp = packet.get_header(Arp) # if is Arp Frame, it doesn't has ip header
        eth = packet.get_header(Ethernet)
        ip = packet.get_header(IPv4)
        in_intf = self.net.interface_by_name(ifaceName)

        mac_target_intf = None
        for intf in self.net.interfaces():
            if intf.ethaddr == eth.dst:
                mac_target_intf = intf
                break

        if(eth.dst != SpecialEthAddr.ETHER_BROADCAST.value and mac_target_intf != in_intf):
            log_info(f"Packet get in wrong interface or has illegal eth.dst , discard")
            return

        if eth.ethertype == EtherType.VLAN:
            log_info(f'Get a VLAN packet, discard')
            return
```

2 Task 3: Forwarding the Packet and ARP

2.1 Coding

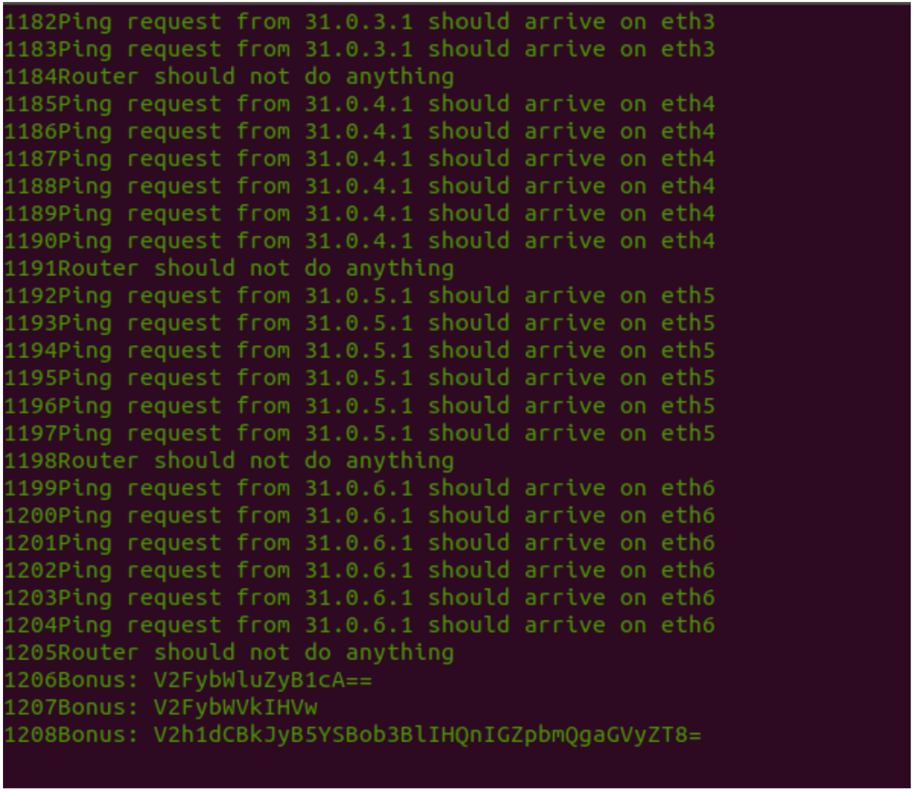
Assignment: In the report, show how you handle packet forwarding and ARP request generation.

Listing 2: packet forwarding and arp request generation

```

def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
        timestamp, ifaceName, packet = recv
        # TODO: your logic here
        #get possible header
        arp = packet.get_header(Arp) # if is Arp Frame, it doesn't has ip header
        eth = packet.get_header(Ethernet)
        ip = packet.get_header(IPv4)
        in_intf = self.net.interface_by_name(ifaceName)

...............

#if is ip datagram
        if ip:
            if ip.dst in self.ip_list:
                log_info(f'get ip packet to {self.net.interface_by_ipaddr(ip.dst).name}, discard')
            else:
                entry =  self.forward_table.match(ip.dst)
                #if forward_table doesn't has ,drop it
                if entry:
                      intf_name = entry['interface_name']
                      intf = self.net.interface_by_name(intf_name)
                      intf_net_address = IPv4Address(int(intf.ipaddr) & int(IPv4Address(intf.netmask)))
                      concate_addr = IPv4Network(f"{intf_net_address}/{intf.netmask}")
                      match = ip.dst in concate_addr
                      # Case 1: If directly reachable
                      # eth_dest is host's mac
                      if match:
                          waiting_pkt = PacketWaitingForMac(packet, ip.dst, intf)
                          self.waiting_packet_update(waiting_pkt)
                      # Case 2: Not directly reachable
                      else:
                          next_hop = entry['next_hop']
                          waiting_pkt = PacketWaitingForMac(packet, next_hop, intf)
                          self.waiting_packet_update(waiting_pkt)

          self.forward_queue()


    def waiting_packet_update(self, pkt):
            has = False
            for ip, info_dic in self.waiting_ip.items():
                if ip == pkt.target_ip:
                    has = True
                    break
            if not has: #yet has packet(with targeted ip) waiting in the list
                dic = {
                    "send_cnt" : 0,
                    "last_send_time" : time.time(),
                    "packets" : [pkt]
                }
                self.waiting_ip[pkt.target_ip] = dic
            else:
                self.waiting_ip[pkt.target_ip]["packets"].append(pkt)

        def forward_queue(self):
            # address the queue in router's main loop
            delete = []

            for ip, dic in self.waiting_ip.items():
                current_time = time.time()
                mac_result = self.arp_table.search(ip)
                if(mac_result): #find ip's accordance
                for waiting_pkt in dic["packets"]:
                    self.forward_ip_packet(waiting_pkt.packet, mac_result, waiting_pkt.interface)
                delete.append(ip)
            elif dic["send_cnt"] < 5:
                if dic["send_cnt"] == 0 or current_time - dic["last_send_time"] > 1.0:
                    #print(f"{ip} send_cnt {dic['send_cnt']}")
                    #send arp request
                    waiting_pkt = dic["packets"][0] #make sure this exist
                    #initialize arguments
                    w_interface = waiting_pkt.interface
                    name = w_interface.name
                    s_mac_addr = w_interface.ethaddr
                    s_ip_addr = w_interface.ipaddr
                    dst_ip_addr = waiting_pkt.target_ip
                    #send arp request
                    arp_request = create_ip_arp_request(s_mac_addr,s_ip_addr,dst_ip_addr)
                    self.net.send_packet(name,arp_request)
                    #update instance
                    dic['last_send_time'] = current_time
                    dic['send_cnt'] += 1
            elif dic["send_cnt"] == 5 and current_time - dic['last_send_time'] > 1.0:

                delete.append(ip)

        for ip in delete:
            #print(f"delete {ip}")
            del self.waiting_ip[ip]

    def forward_ip_packet(self,packet,eth_dest,interface):
        # modify eth header
        eth_header = packet.get_header(Ethernet)
        eth_header.src = interface.ethaddr
        eth_header.dst = eth_dest
        packet[IPv4].ttl -= 1
        #send
        self.net.send_packet(interface.name,packet)
```

2.2 Testing

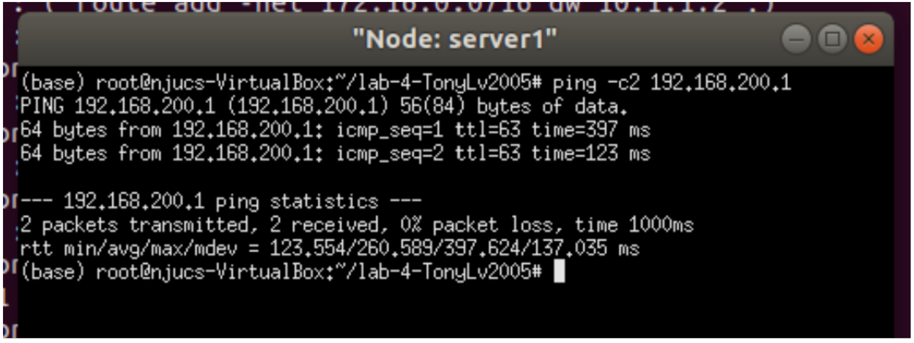
Figure 1: Test Result



```
1182Ping request from 31.0.3.1 should arrive on eth3
1183Ping request from 31.0.3.1 should arrive on eth3
1184Router should not do anything
1185Ping request from 31.0.4.1 should arrive on eth4
1186Ping request from 31.0.4.1 should arrive on eth4
1187Ping request from 31.0.4.1 should arrive on eth4
1188Ping request from 31.0.4.1 should arrive on eth4
1189Ping request from 31.0.4.1 should arrive on eth4
1190Ping request from 31.0.4.1 should arrive on eth4
1191Router should not do anything
1192Ping request from 31.0.5.1 should arrive on eth5
1193Ping request from 31.0.5.1 should arrive on eth5
1194Ping request from 31.0.5.1 should arrive on eth5
1195Ping request from 31.0.5.1 should arrive on eth5
1196Ping request from 31.0.5.1 should arrive on eth5
1197Ping request from 31.0.5.1 should arrive on eth5
1198Router should not do anything
1199Ping request from 31.0.6.1 should arrive on eth6
1200Ping request from 31.0.6.1 should arrive on eth6
1201Ping request from 31.0.6.1 should arrive on eth6
1202Ping request from 31.0.6.1 should arrive on eth6
1203Ping request from 31.0.6.1 should arrive on eth6
1204Ping request from 31.0.6.1 should arrive on eth6
1205Router should not do anything
1206Bonus: V2FybWluZyB1cA==
1207Bonus: V2FybWVkIHVw
1208Bonus: V2h1dCBkJyB5YSBob3BlIHQnIGZpbmQgaGVyZT8=
```

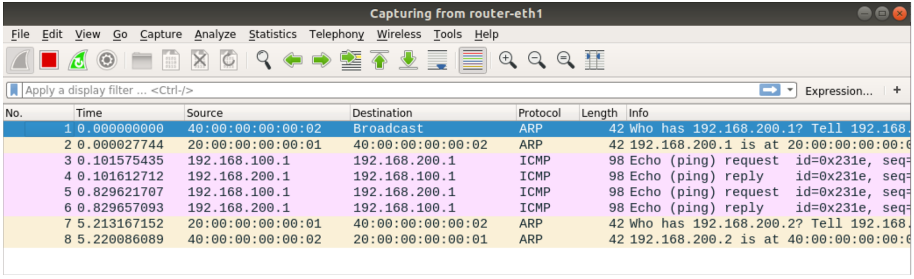
2.3 Deploying

Figure 2: Server1 Ping -c2 Server2



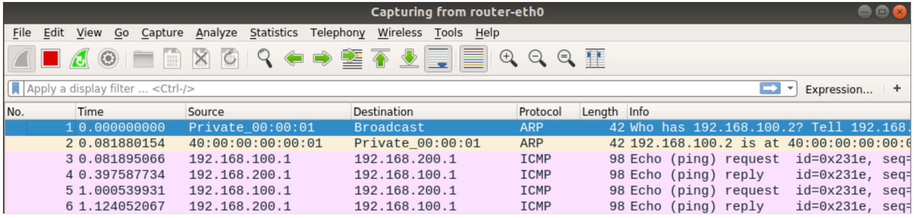
```
(base) root@njucs-VirtualBox:~/lab-4-TonyLv2005# ping -c2 192.168.200.1
PING 192.168.200.1 (192.168.200.1) 56(84) bytes of data.
64 bytes from 192.168.200.1: icmp_seq=1 ttl=63 time=397 ms
64 bytes from 192.168.200.1: icmp_seq=2 ttl=63 time=123 ms

--- 192.168.200.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 123.554/260.589/397.624/137.035 ms
(base) root@njucs-VirtualBox:~/lab-4-TonyLv2005# 
```

Figure 3: Capture of router-eth1



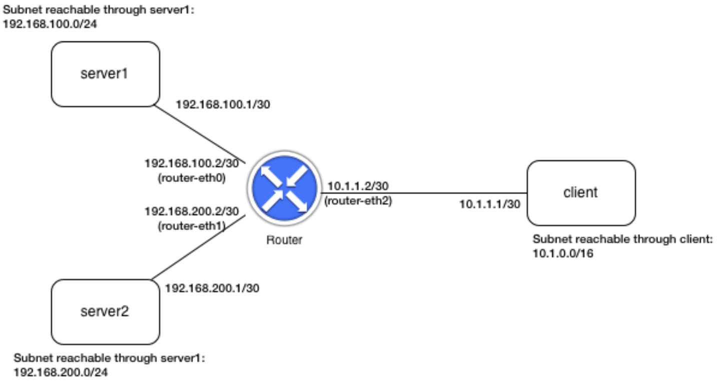
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	40:00:00:00:00:02	Broadcast	ARP	42	Who has 192.168.200.1? Tell 192.168.
2	0.000027744	20:00:00:00:00:01	40:00:00:00:00:02	ARP	42	192.168.200.1 is at 20:00:00:00:00:0
3	0.101575435	192.168.100.1	192.168.200.1	ICMP	98	Echo (ping) request id=0x231e, seq=
4	0.101612712	192.168.200.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x231e, seq=
5	0.829621707	192.168.100.1	192.168.200.1	ICMP	98	Echo (ping) request id=0x231e, seq=
6	0.829657093	192.168.200.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x231e, seq=
7	5.213167152	20:00:00:00:00:01	40:00:00:00:00:02	ARP	42	Who has 192.168.200.2? Tell 192.168.
8	5.220086089	40:00:00:00:00:02	20:00:00:00:00:01	ARP	42	192.168.200.2 is at 40:00:00:00:00:0

Figure 4: Capture of router-eth0



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Private_00:00:01	Broadcast	ARP	42	Who has 192.168.100.2? Tell 192.168.
2	0.081880154	40:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.2 is at 40:00:00:00:00:0
3	0.081895066	192.168.100.1	192.168.200.1	ICMP	98	Echo (ping) request id=0x231e, seq=
4	0.397587734	192.168.200.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x231e, seq=
5	1.000539931	192.168.100.1	192.168.200.1	ICMP	98	Echo (ping) request id=0x231e, seq=
6	1.124052067	192.168.200.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x231e, seq=

Figure 5: Network Topology



server1 ping server2的过程中，先向eth0接口发送arp request,请求接口的mac地址，router在eth0以arp reply回应，然后发送ICMP包。Router此时对从接口eth1输入的ICMP包进行Forwarding操作，先将它们放入arp_waiting_ip下属的packets里面，并且加入192.168.200.1这个ip进行发送arp，所以从eth1接口向server2发送arp request,接受 arp_reply从而得到了server2的mac地址放入arp表，然后将waiting_ip里面的packets全都发送并清空，所以在eth1接口中向server2发送了先前进入router的icmp包，完成转发